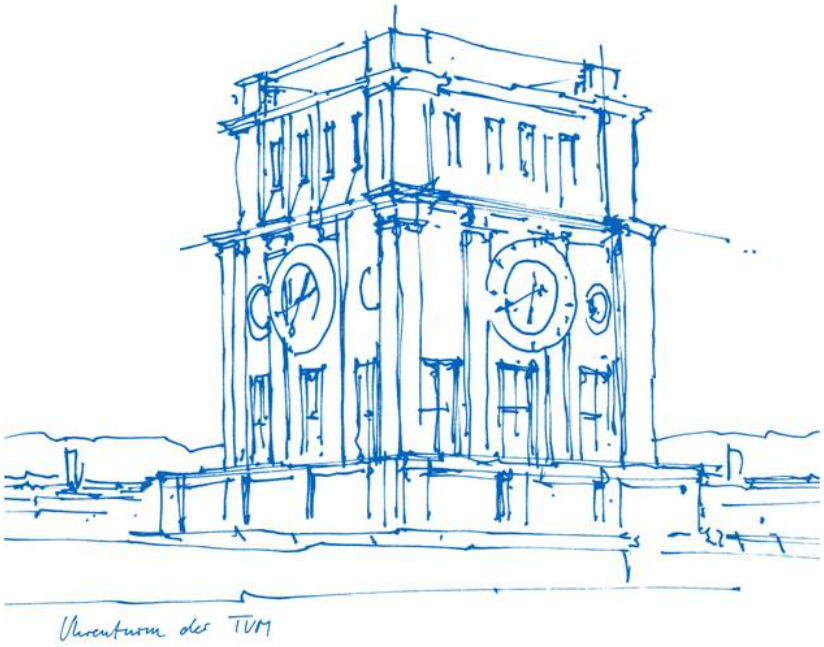
Exercises for
Social Gaming and
Social Computing
(IN2241 + IN0040) – Introduction to

Exercise Sheet 4

Sentiment Analysis in DotA



Exercise Content

1. Introduction to Python and Network Visualization — **June 6-12**

Python basics and first look at network analysis and visualization with the Simpsons and NetworkX

2. Social Network Analysis: Comparison of Centrality Measures — **June 13-19**

Deepen knowledge about centrality measures by comparing different measures using NetworkX and your own implementations.

3. Social Recommender Systems — **June 20-26**

Investigate variations of the famous Collaborative Filtering algorithm for recommending games using data from Steam.

4. Sentiment Analysis on DotA with Linear Regression — **June 27-July 3**

Analyze data from the game DotA in terms of sentiment.

5. Clustering: Identifying Player Types in WoW — **July 4-10**

Using data from WoW we aim at comparing different clustering paradigms (etc. GMMs or DBSCAN) for identifying player types

6. NLP on Social Media: Climate Change Activists vs. Deniers — **July 11-17**

With the help of modern NLP techniques and deep learning models, we analyze a Twitter dataset on Climate change

Exercise Sheet 4: Sentiment Analysis in DotA

- goal: analyse DotA chat logs and use sentiment analysis to find out whether a negative attitude affects a player's win rate

- DotA
  - two teams of 5 players (that usually don't know each other)
  - each team tries to destroy the enemy base
  - the nature of the game does provoke negativity at times
  - and we want to try to predict it

The Data: Your friends' Steam library

- information from 1.500 matches played during December 2016, split in the following files:
  - *chat.csv*: what was said in the chat between teams, when it was said and which player said it
  - *match.csv:* detailed statistics for every match
  - *players.csv:* detailed statistics for every player
  - *player_times.csv:* gold & exp. for every player each minute of every game
  - *player_ratings.csv*: An estimation of a players skill based on his lifetime winrate.
  - *labels.csv:* a sample of labeled chat used for the sentiment analysis

Task 4.1: Your first sentiment analysis

Sentiment analysis, sometimes called opinion mining, is a method to derive information from the text that allows for a classification as neutral, positive or negative. It is a semi-supervised process, meaning that you need a small set of labeled data to train your machine learning model on in order to use it on another set of unlabeled data. Without the labeled set it would be difficult for your AI to know what exactly makes a statement positive or negative. Some of its many practical applications are the analysis of customer reviews, social media comments or survey responses.

Note: We will use a random forest classifier in this task.

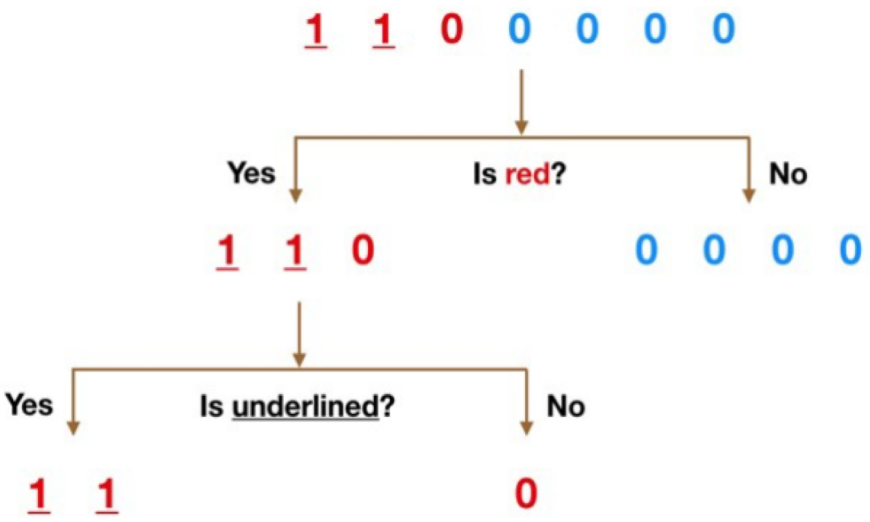
Your task is to train a model using the labeled data, then use that model to predict the sentiments of the whole chat. Let us start with the basics:

Sentiment Analysis

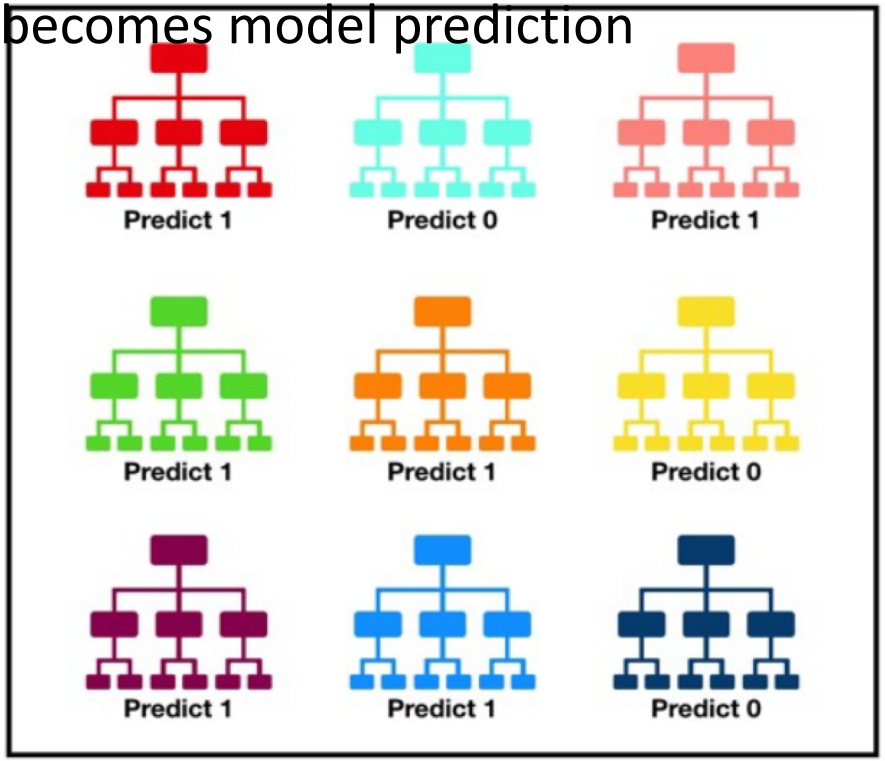
- input: text (e.g. tweets, comments, reviews)
- goal: classify sentiment of the text (usually neutral, negative or positive)
- means: machine learning model
- requirements: pre-labeled text (called “train-set”).
  - labels.xlsx in our case

Sentiment Analysis: Machine Learning

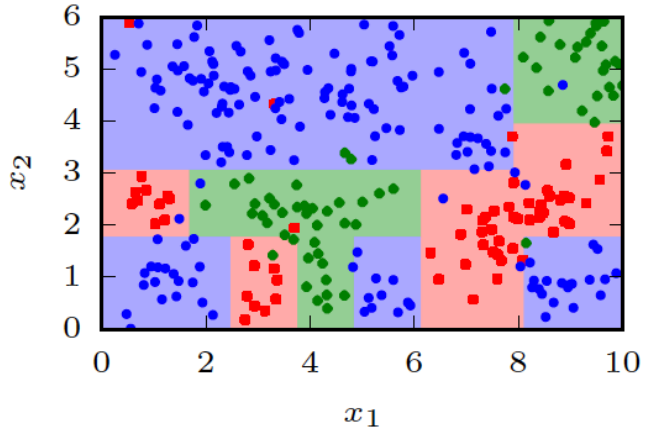
- machine learning algorithm: random forest classifier
- general idea:
  - build decision trees for classification
  - every tree makes one prediction
  - the class with most predictions becomes model prediction

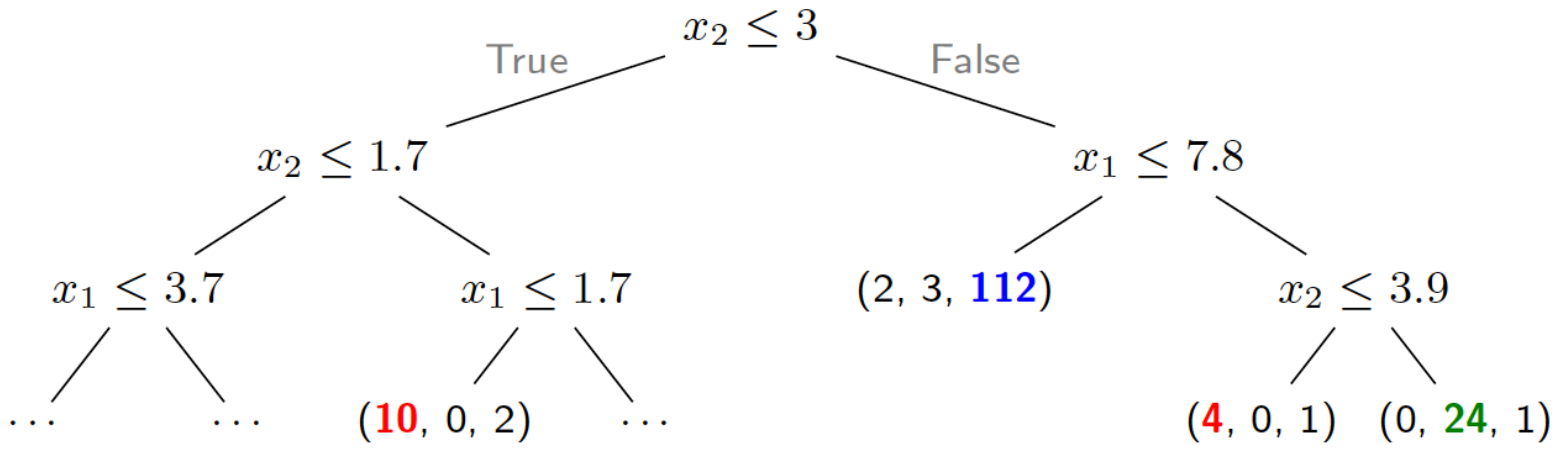


Simple Decision Tree Example



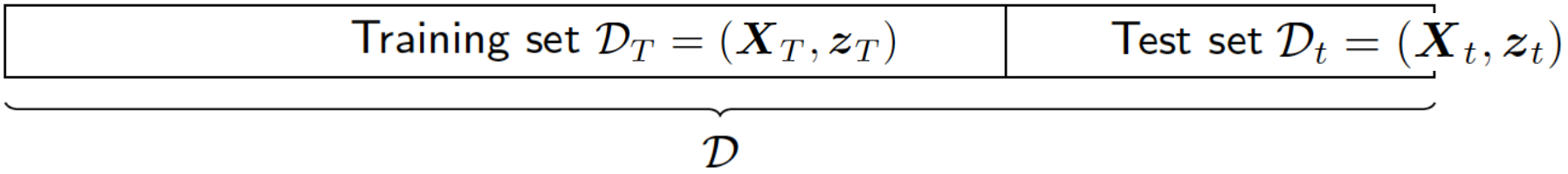
Decision Trees [2], [5]





Distribution of classes in leaf: (red, green, blue)

Decision Trees [2], [5]



- Idea: Build all possible trees and evaluate their performance on $\mathcal{D}_t$
- All concrete features and values in $\mathcal{D}_T$ can serve as tests in the tree.
- Sadly: „all possible trees“ → combinatorical explosion → finding optimal tree is NP-complete
- → Grow the tree top-down and choose the best split node-by-node using a greedy heuristic on the training data

feature	tests
x_1	$\leq$ 0.36457631
	$\leq$ 0.50120369
	$\leq$ 0.54139549
	$\leq \ldots$
x_2	$\leq$ 0.09652214
	$\leq$ 0.20923062
	$\leq \ldots$

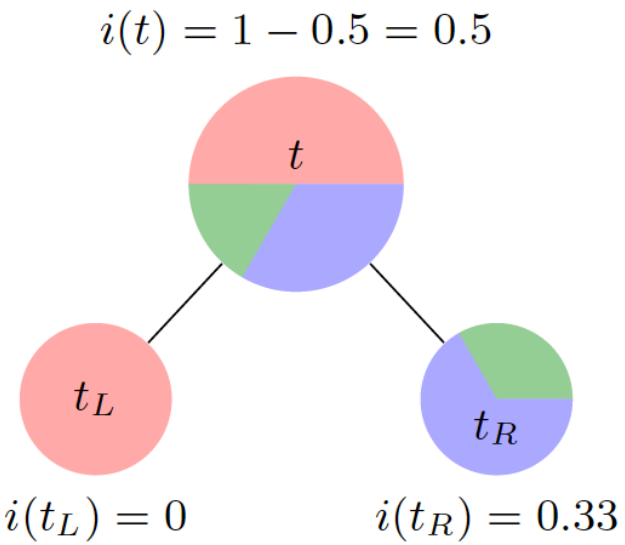
Decision Trees [2], [5]

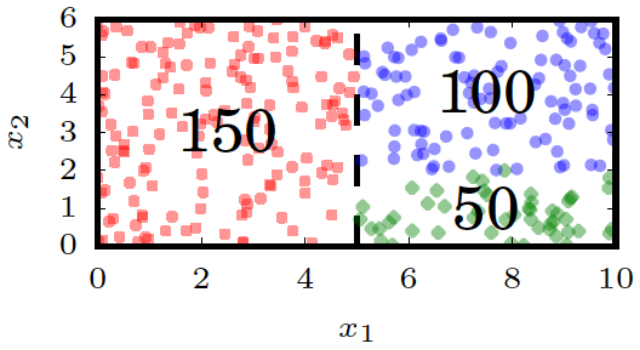
For example: Split the node when it improves the missclassification rate i_E at node t

$$i_E(t) = 1 - \max_c \; p(z = c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is given by

$$\Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R)$$





Impurity measures

With $\pi_c = p(z = c \mid t)$:

Misclassification rate:

$$i_E(t) = 1 - \max_c \ \pi_c$$

Entropy:

$$i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log \pi_{c_i}$$

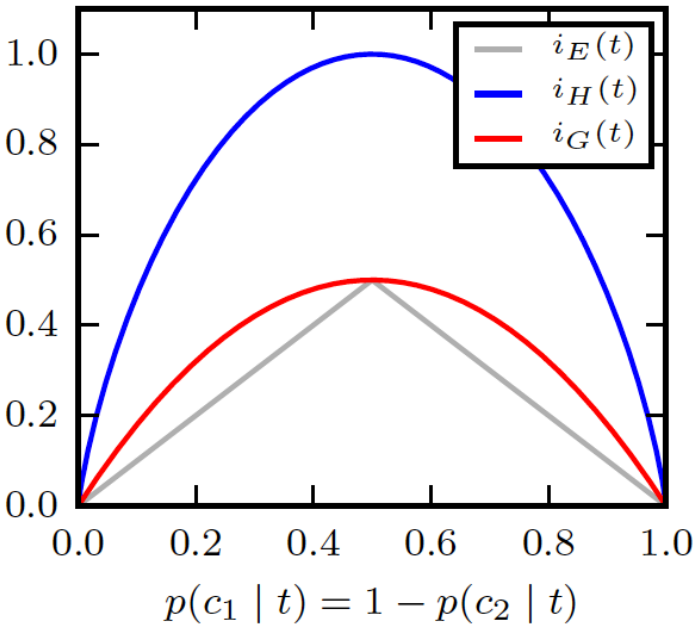
(Note that $\lim_{x \to 0+} x \log x = 0$.)

Gini index:

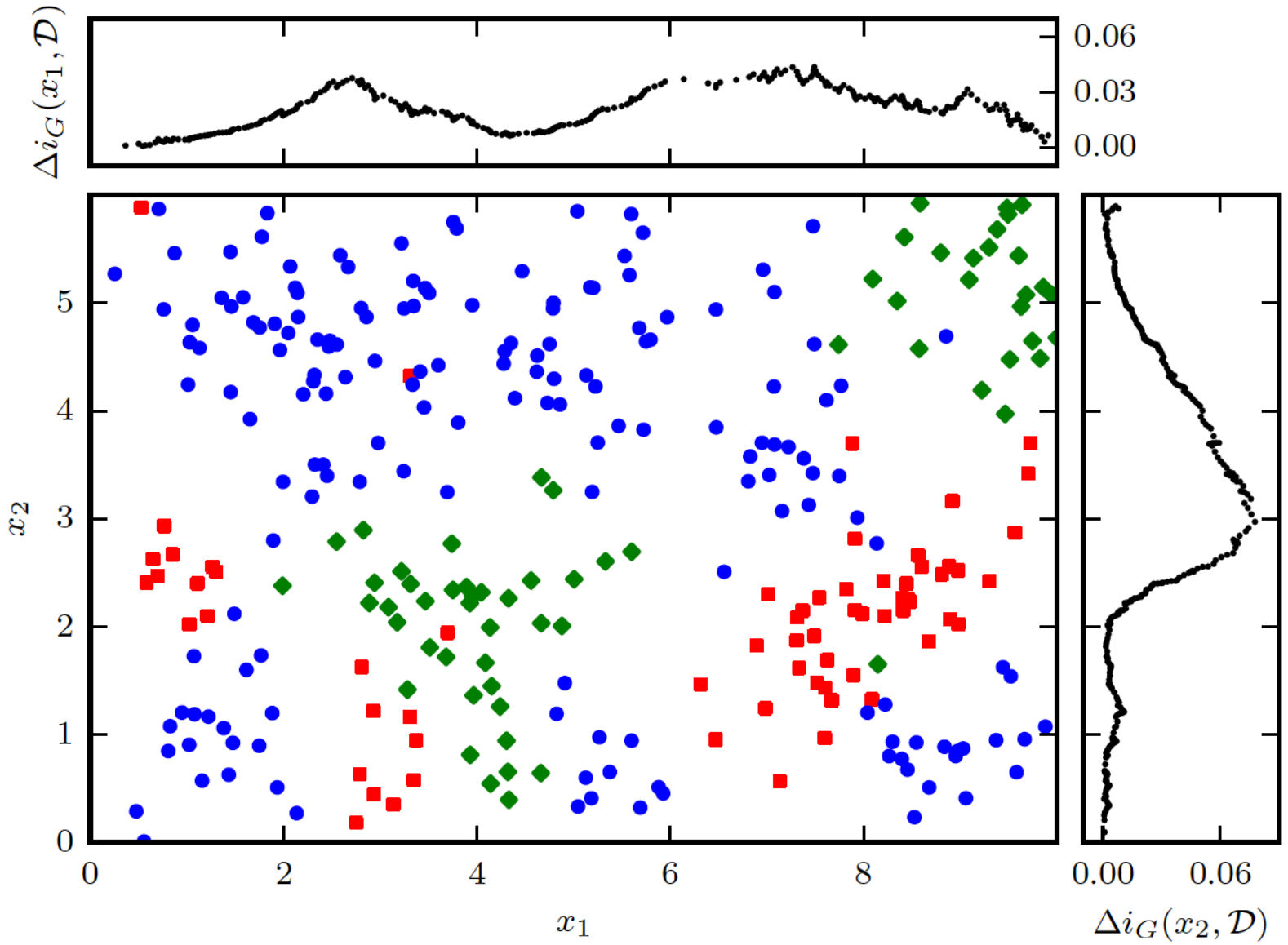
$$i_G(t) = \sum_{c_i \in C} \pi_{c_i}(1 - \pi_{c_i})$$

$$= 1 - \sum_{c_i \in C} \pi_{c_i}^2$$

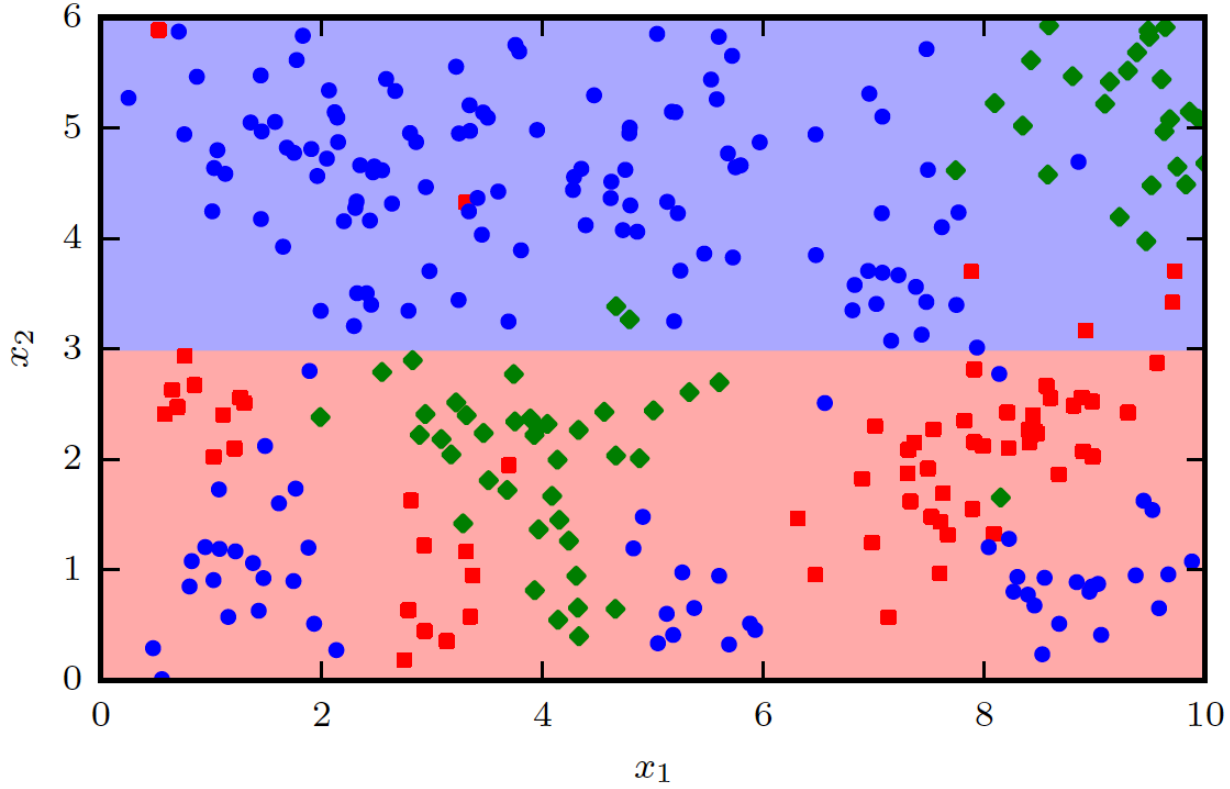
For $C = \{c_1, c_2\}$:



Decision Trees [2], [5]

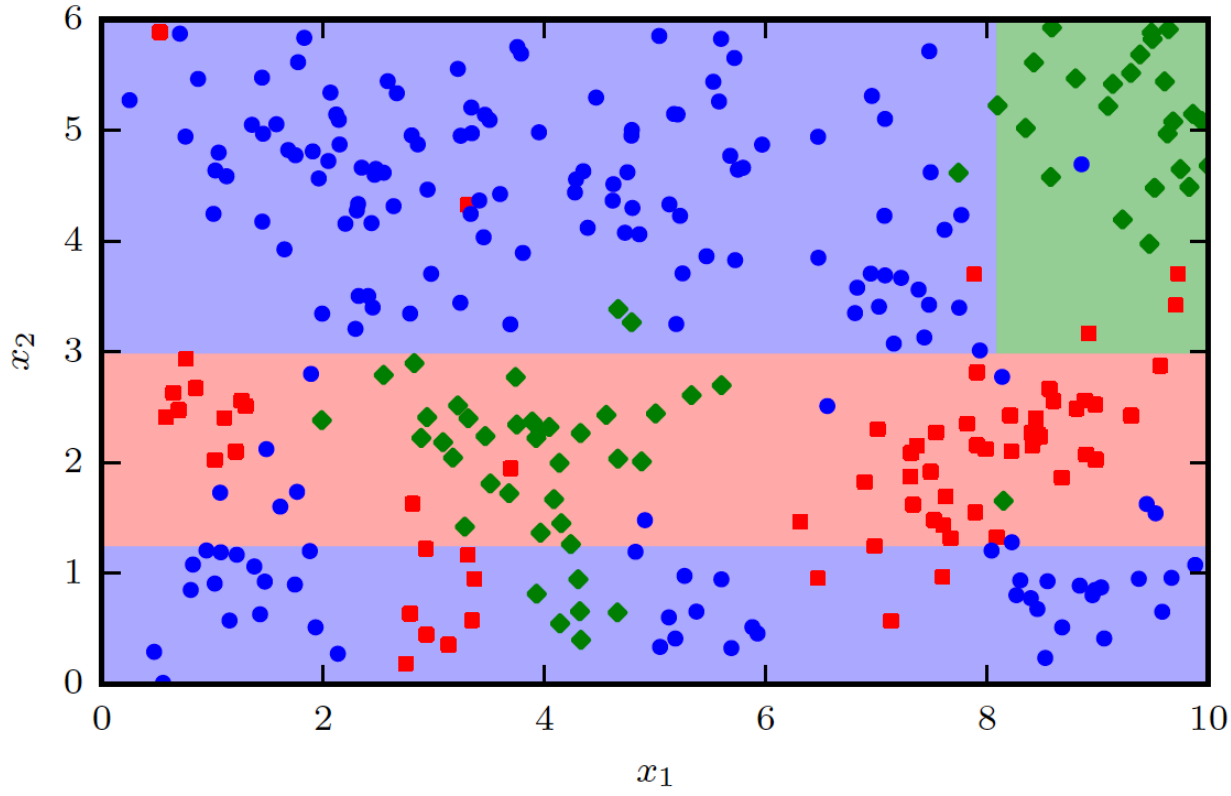


Decision boundaries at depth 1



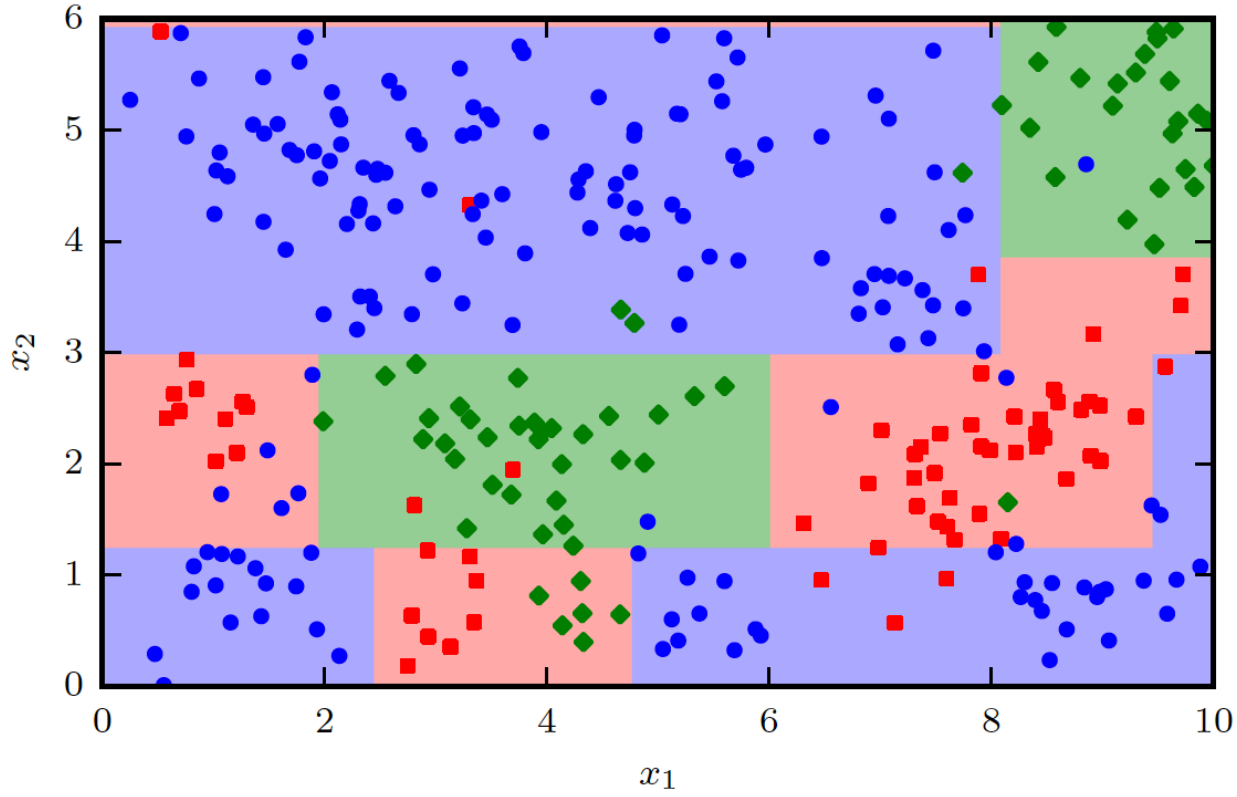
Accuracy on the whole data set: 58.3%

Decision boundaries at depth 2



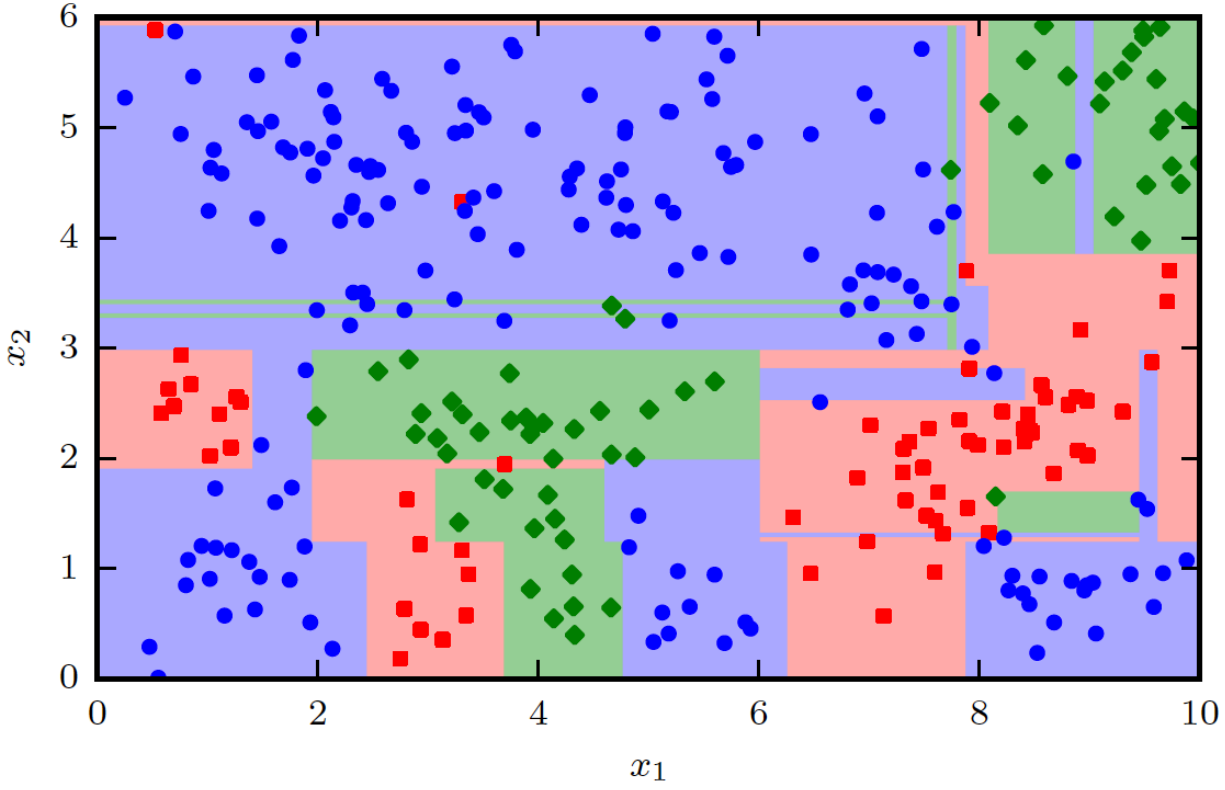
Accuracy on the whole data set: 77%

Decision boundaries at depth 4



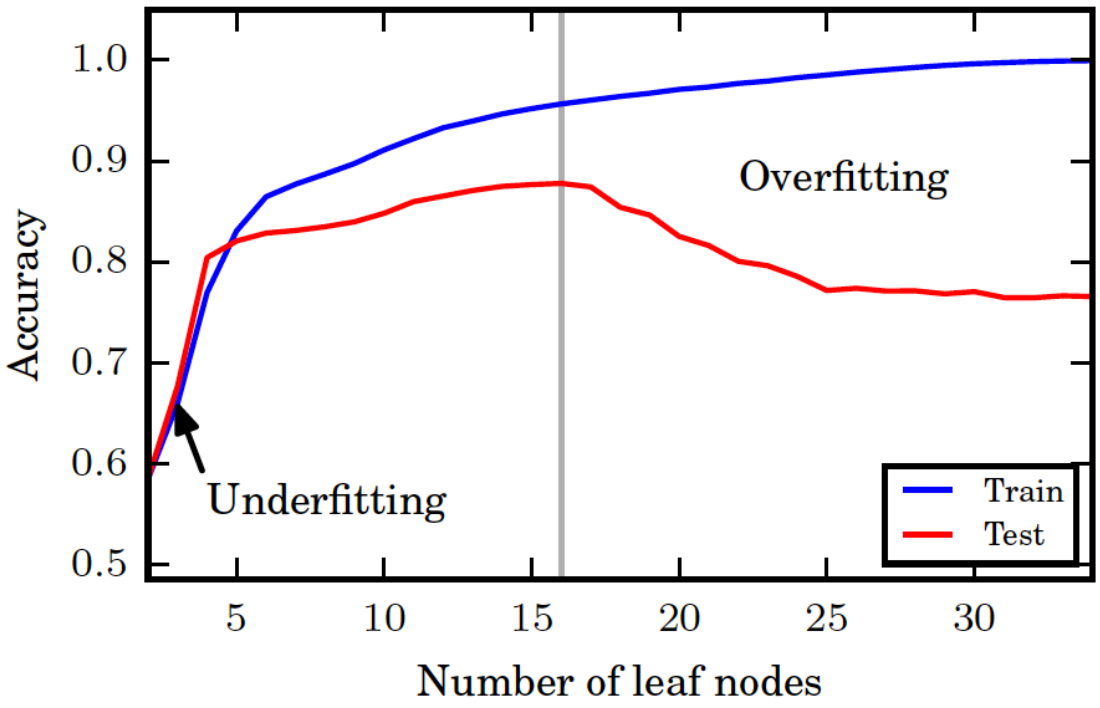
Accuracy on the whole data set: 90.3%

Decision boundaries of a maximally pure tree



Accuracy on the whole data set: 100% $\rightarrow$ *generalisation*?

Decision Trees [2], [5]



Ideas to counter overfitting:

- limit depth when growing tree
- stop growing when splitting benefit drops below certain threshold
- grow maximally and prune back afterwards

Random Forests: Introducing Variance

Let N denote the number of samples in the training set $\mathcal{D}_T$ and D denote the number of features (attributes, dimensions).

- Instead of using the whole training set to build the tree, we build each tree T_i from a bootstrap sample:
  From $\mathcal{D}_T$ create $\mathcal{D}_{T_i}$ by *randomly* drawing N samples *with replacement*.

  By sampling with replacement we only use ≈ 63.2 % of $\mathcal{D}_T$ for each tree.
- Build decision trees T_i top-down with a greedy heuristic and a small variation:
  At each node, instead of picking the best of all possible tests, *randomly* select a subset of $d < D$ features and consider only tests on these features.
- There is no need for pruning.

Simple TF-IDF Features for Text

- What kinds of features can we use if the input data is text?

  → -dimensional tf-idf vectors:

"I like to dance samba, bake pizza, watch tv and plant trees in the garden. I also like to bake cakes."

I	2
like	2
to	2
dance	1
samba	1
bake	2
pizza	1
watch	1
tv	1
and	1
plant	1
trees	1
in	1
the	1
garden	1
also	1
cakes	1

*Often: Instead of term-frequency (tf) alone: use term-frequency * inverse document frequency (idf); idf = - log (#of docs where t occurs / #of docs)*

Sentiment Analysis

- first, the model needs to be trained:
  - feed the labeled data into the classifier
  - split the data into train-set and test-set
  - train on the train-set
  - report performance with test-set
- the trained model is now ready to be used on unlabeled data

Tasks (cont.)

a) Preparation

Import the labels and split them into two arrays: the chat itself and the labels.

A label is like a review of a single message:
-1 = negative
0 = neutral
1 = positive

b) How to train your model

In this step you will use the chat and labels to train your random forest classifier. In order to do so, create the random forest classifier, fit it and make a prediction on the test set.

After you are done, print the accuracy score and comment on it.

Hints:

- When creating the classifier, use n_estimators=200, random_state=0 as arguments.
- The test should be 20% of the whole set

Tasks (cont.)

c) Prediction time:

Now you can use the model to **predict the sentiments** for the whole chat. Import the chat and **predict the labels**. You will need to use `vectorizer.transform().toarray()` on your data, but **DO NOT** use `fit()` anywhere! The classifier is already fitted, fitting it again effectively erases all it has learned.

Note: The chat table is massive. Labelling all of it may take a while.

In [ ]:

```
chatData = pd.read_csv("chat.csv")
unlabeled = chatData.iloc[:,1].values

# We remove the first 475 entries as they are the ones contained in our labeled training set,
# so we will rather use their original hand made labels.
unlabeled = unlabeled[475:]

# TODO:

```

Linear Regression: Theory

- is a technique that fits a function from input variables X to a dependent variable y:

$$y = \alpha + \beta X + \epsilon$$

  - training data:
  - X is the predictive vector, containing the (predictive) variables
  - α and β are the model's parameters, where
    - α is the intercept/bias
    - β the coefficient vector containing coefficients for each predictive variable
  - And ϵ is an assumed Gaussian error

Linear Regression: Theory

- is a technique that tries to find a correlation between a set of input variables X and a dependent variable y:

- our goal:
  - input (gold advantage of teams, worst kill-death ratio per team, negativity) as X
  - determine whether = radiant_win can be modeled as a function of the other factors

Linear Regression: Model

$$y = \alpha + \beta X + \epsilon \qquad \alpha \in \mathbb{R},\ \boldsymbol{\beta} \in \mathbb{R}^{d-1},\ \boldsymbol{X} \in \mathbb{R}^{d-1}$$

$$y = \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x} \qquad \boldsymbol{w}^{\mathbf{T}} = (\alpha, \beta_1, \beta_2, \dots, \beta_{d-1}) \in \mathbb{R}^d;$$
$$\boldsymbol{x} = (1, X_1, X_2, \dots, X_{d-1}) \in \mathbb{R}^d$$

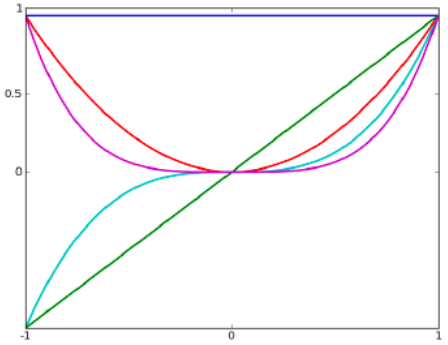
Or using feature map $\phi: \mathbb{R}^n \overset{y^{(1)}}{\rightarrow} \mathbb{R}^m$:

$$y = \boldsymbol{w}^{\mathbf{T}}\boldsymbol{\phi}(\boldsymbol{x}) \qquad \boldsymbol{w}^{\mathbf{T}} \in \mathbb{R}^m \text{ } \textit{(using same symbol } \boldsymbol{w} \textit{ for notational simplicity)}$$
$$\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \dots, \phi_m(\boldsymbol{x})) \in \mathbb{R}^m$$
$$\boldsymbol{x} \in \mathbb{R}^d$$

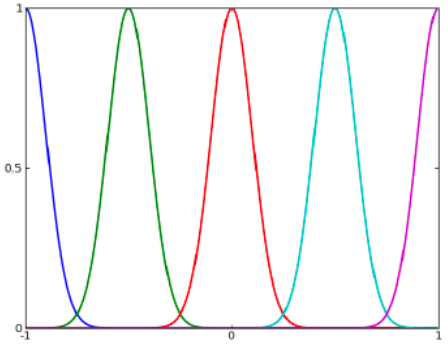
Linear Regression: Feature Maps

Examples for ϕ (using *n=1*)

Polynomials $\phi_j(x) = x^j$

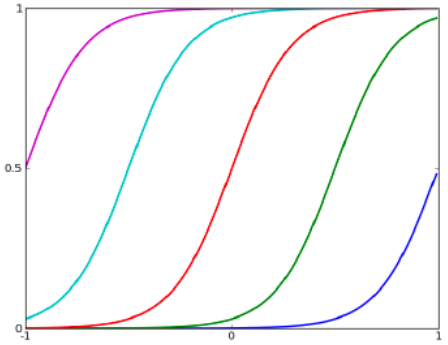


Gaussian $\phi_j(x) = \mathrm{e}^{\frac{-(x-\mu_j)^2}{2s^2}}$



Logistic Sigmoid $\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$,

where $\sigma(a) = \frac{1}{1+\mathrm{e}^{-a}}$



Linear Regression: Training / Fitting

$$y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

Training-Dataset $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i\in[1,N]}$

using simple quadric error error function:

$$\boldsymbol{w}^* = argmin_{\boldsymbol{w}} \sum_{i=1}^{N} \left(y^{(i)} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}^{(i)})\right)^2$$

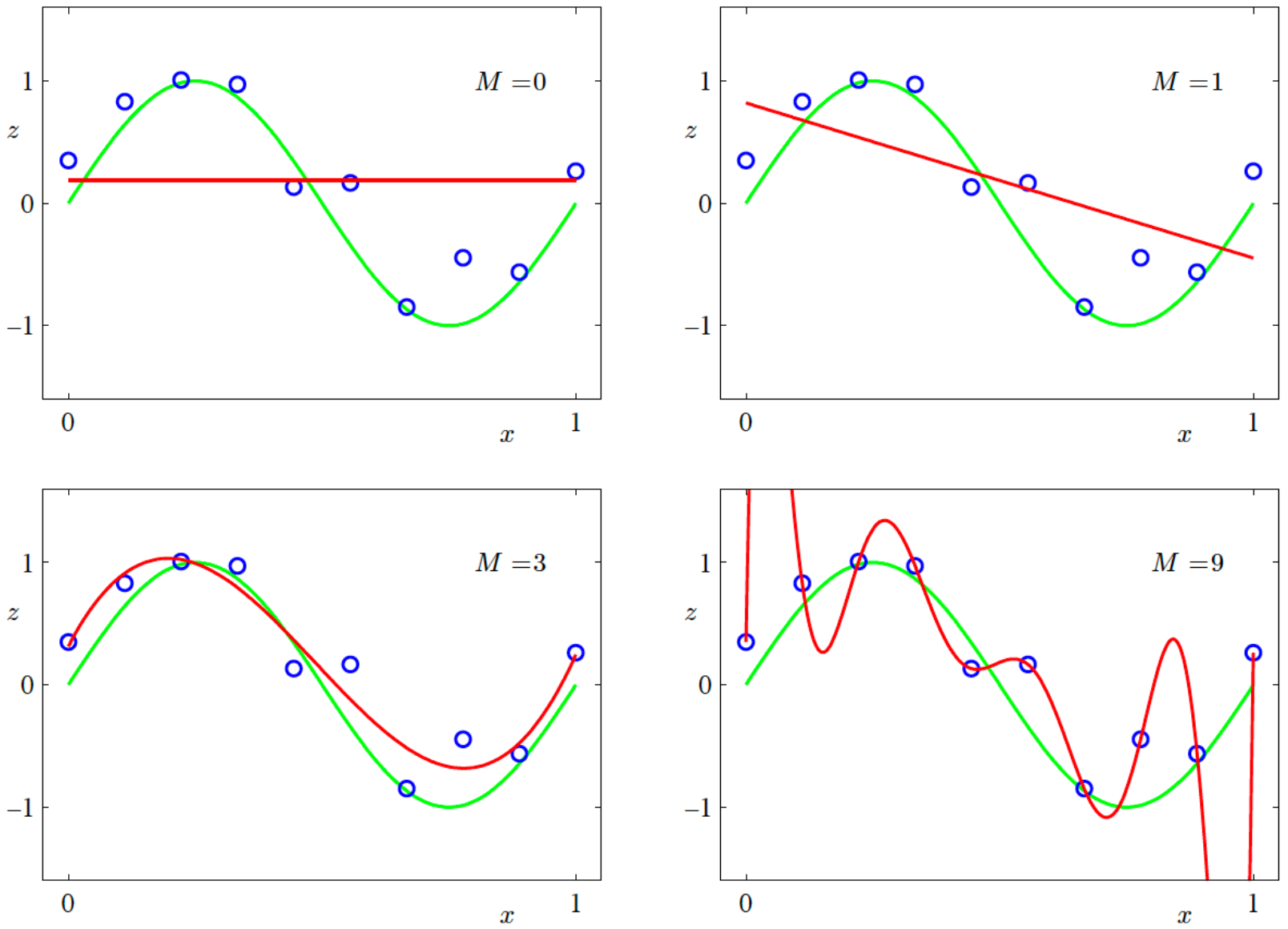
computing gradient, setting to zero yields:

$$w^* = (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

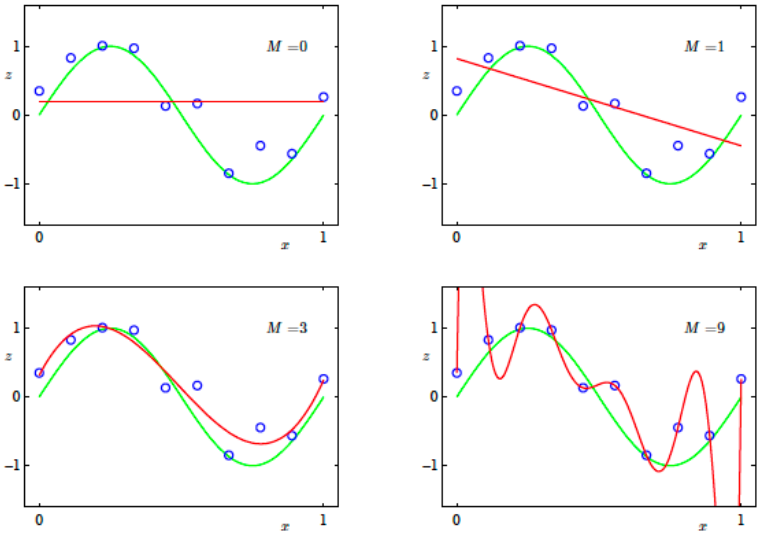
$$\boldsymbol{y} = (y^{(1)}, y^{(2)}, \dots, y^{(N)}) \in \mathbb{R}^N$$

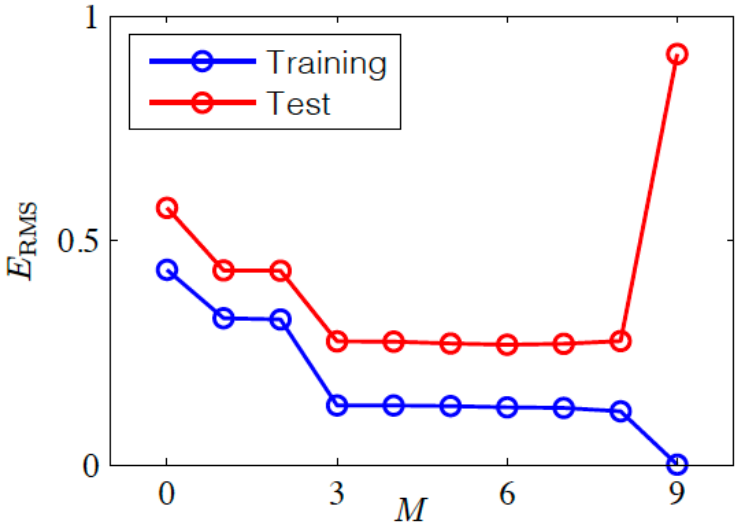
$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_1(\boldsymbol{x}^{(1)}) & \phi_2(\boldsymbol{x}^{(1)}) & \dots & \phi_m(\boldsymbol{x}^{(1)}) \\ \phi_1(\boldsymbol{x}^{(2)}) & \phi_2(\boldsymbol{x}^{(2)}) & \dots & \phi_m(\boldsymbol{x}^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\boldsymbol{x}^{(N)}) & \phi_2(\boldsymbol{x}^{(N)}) & \dots & \phi_m(\boldsymbol{x}^{(N)}) \end{pmatrix} \in \mathbb{R}^{N\times m}$$

Linear Regression: Polynomial Feature Maps: choosing m



Linear Regression: Polynomial Feature Maps: choosing m





One valid solution is to choose M using the standard train-validation split approach.

Linear Regression: Polynomial Feature Maps: choosing m

Other approach: change error function: penalize large values of $\boldsymbol{w}$ (characteristic for overfitting case): Ridge Regression

$$\boldsymbol{w}^* = argmin_{\boldsymbol{w}} \sum_{i=1}^{N} \left(y^{(i)} - \boldsymbol{w}^{\mathbf{T}} \boldsymbol{\phi}\left(\boldsymbol{x}^{(\boldsymbol{i})}\right) \right)^2 + \lambda \left|\left|\boldsymbol{w}\right|\right|^2$$

or Lasso Regression

$$\boldsymbol{w}^* = argmin_{\boldsymbol{w}} \sum_{i=1}^{N} \left(y^{(i)} - \boldsymbol{w}^{\mathbf{T}} \boldsymbol{\phi}\left(\boldsymbol{x}^{(\boldsymbol{i})}\right) \right)^2 + \lambda \left|\left|\boldsymbol{w}\right|\right|_{L1}$$

Tasks (cont.)

a) Preparation:

1. First, we need to **read the .csv files** and group them by `match_id`.

2. **Create a dataframe** containing all relevant information where each row represents one match. An empty dataframe has already been created for you with all the columns you need to fill. Fill the columns with the following elements:

3. **Create a list of lists of lists** called `full_chatdata`, with a tuple for each message, with label and team (you can see an example in the comments).

4. **Create a list** called `goldData` containing the gold advantage for every timestamp of a match (usually every minute).

5. **Create a list** called `skillTeams` of the skill values for each player in a match, split into two parts, one for each team, called `skillRadiant` and `skillDire`.

6. **Add an additional column** called `radiant_win` displaying the winning team with a boolean value.

Hints:

- you can find the labels in the chatData dataframe
- use the `slot` column to determine the team. 0 to 4 is for radiant, 5-9 is for dire
- There is a column in the match.csv file called `radiant_win` that displays true if team radiant won, false if not

Tasks (cont.)

b) Per-match analysis:

As you may have noticed, the gold variables are gathered every minute, but the chat times are irregular. We could try to group the chat into 60 second timeframes that would correspond with the gold values, but this would be too tedious. Instead, we will simplify this by looking at the game as a whole:

1. **Compute the average negativity** for each team by iterating over the list of tuples you created in exercise 4.2.3.

2. Then, **compute the average gold advantage** for each match, and add a column for the gold advantage at the end of a match. The gold advantage at the end of a match is the last value of the list.

3. **Create a new column** for the difference in negativity between the two teams.

4. The skill values aren't very useful in the current format. Take the difference of the lowest and highest skilled player from each team and **create new columns** for the difference. The reasoning behind this is that a high skill difference in a team probably leads to more harassment.

Tasks (cont.)

c) A warm-up regression

Before we test our hypothesis of whether or not the state of the game influences player behavior, we will perform a linear regression with only one input variable: The gold advantage.

You have probably wondered why we just assume that the gold values would represent the state of a game, whether a team is losing or winning. So far, this has only been a theory, and we should test it, as it would not make sense to use it as a representation for the state of the game in the actual regression model, if it wasn't representative at all.

1. Once again, split your data into a train set and test set, create a linear regression model, fit the data and print your score. Try it two times: Your dependant variable should always be `radiant_win`, your X should be the average gold advantage and the gold advantage at the end.
2. Discuss the score you obtained! What do the results mean for the explanatory power of the gold variables?

Tasks (cont.)

d) Testing our hypothesis

Finally, we can do our linear regression. This time, **use the newly created columns from 4.2 b)**. Use the **gold-related data** and the **diffrence in skill for X**, and the **difference in negativity for Y**.

```
X = # TODO: gold-related columns, skill-related columns
Y = # TODO: difference in negativity

# with sklearn
regr = linear_model.LinearRegression()
regr.fit(X, Y)
print(regr.score(X,Y))
```

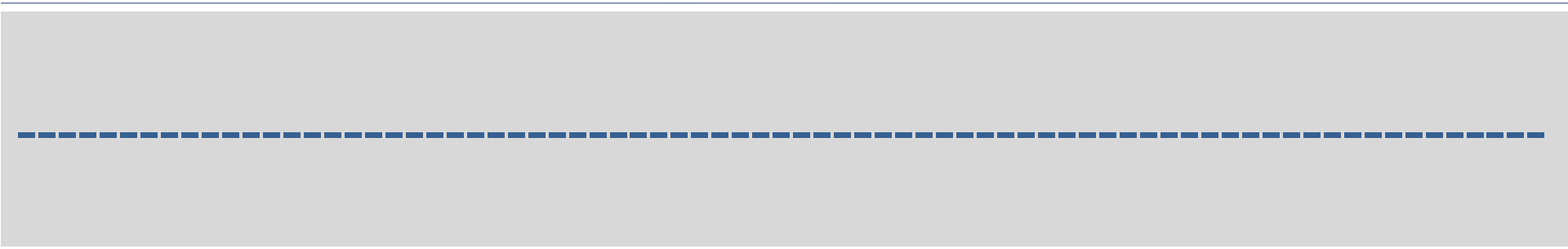
e) Discussion

What is the score? What does that number mean? Discuss possible reasons for this result.
Hint: Take a peek at the labels.csv file and look at some of the most common negative words.

TODO: Write your observations here:

Submitting your solution

- work by expanding the .ipynb iPython notebook for the exercise that you downloaded from Moodle
- save your expanded .ipynb iPython notebook in your working directory
- submit your .ipynb iPython notebook via Moodle (nothing else)
- remember: working in groups is not permitted. Each student must submit their own .ipynb notebook!
- we check for plagiarism. Each detected case will be graded with 5.0 for the whole exercise
- deadline: check Moodle



Citations

[1] E. Gilbert and K. Karahalios: *Predicting Tie Strength With Social Media*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2009.

[2] C. Bishop: *Pattern Recognition and Machine Learning*. 2006, chapter 3

[3] Lecture Machine Learning I, TUM Informatics (Winter 2017)

[4] https://degninou.net/2016/02/04/multiple-regression-and-diagnostics-with-python/

[5] Lecture Machine Learning I, TUM Informatics (Winter 2016)